Validating Your PSQL Database Backups

A White Paper From

For more information, see our web site at
http://www.goldstarsoftware.com

Validating Your PSQL Database Backups

Last Updated: 09/15/2016

Do you have a mission critical database environment? Do you have one of those new-fangled hybrid/cloud backup solutions? Is your PSQL database data REALLY protected from disasters, rogue employees, and malware attacks?

A Bedtime Story, or Perhaps a Nightmare

Recently, we were tasked with helping a client recover data from their mission-critical database which had been rendered useless by a Cryptolocker-style, data-encrypting, malware attack. Normally, this would be no big deal, as we would recommend that they just restore from the latest full backup, take the loss of data involved, and go on about the day normally. However, this was not going to be a normal day.

This client had recently moved to a new virtual server, and they had also implemented a brand new hybrid/cloud backup solution. This particular solution was configured to scan their server for data changes every hour, take a VSS snapshot to get the data into a consistent state, and then copy the latest changes to a local data repository. After the hourly snapshot completed, the local device pushed the changed data to the cloud, providing a complete disaster-recovery solution, just in case the entire data center melted down. With local data on the repository, local restores could be done very quickly, and with the data in the cloud, a new virtual machine could be stood up in the cloud data center, recovering from a full-fledged disaster in minutes. What a great idea!

What they had *not* known is that the backup solution was NOT capturing every disk write. Instead, it was ONLY looking at changes to the size and timestamp of each file (often called an *incremental backup*), and then using *that* information to decide if a file needed to be backed up. It then took the changed data files and applied them to the original "full" backup of the server on the local device (and in the cloud), and then reported to have successfully built a full, restorable copy of the server. This solution was silently doing its job every hour of every day, typically backing up 30-40MB of data each hour, and nobody questioned it.

While a backup like this would have been fine for a gaggle of spreadsheets and other such documents, the Actian PSQL database required by their mission-critical application is a very different animal. In order to optimize performance, PSQL writes directly to the database files themselves, bypassing the OS cache. The net result is that the file timestamps are ONLY updated when the file is completely closed by the database engine, meaning that ALL users are done using that database file. While this wouldn't be an issue for a typical application used only during the day (as all files would be closed at night and thus backed up nightly), a mission-critical system can be accessed by users for weeks or months at a time.

So, what happens when these two worlds collide? Each PSQL table timestamp is updated ONLY when all users are out of the application. For some files, this means that the changes are immediately written, the timestamp is updated, and the file is backed up the next hour. Other files were only closed at the end of the day, and therefore they were only backed up during the LAST backup of the day. Indeed, for this environment, some of the files were NEVER closed, and thus were never backed up by the incremental backup process. This meant that the only valid backup of these files was from immediately after the last server reboot – some 5 weeks earlier!

If you know anything about a relational database, you can already see where this is heading. After the malware attack, they opted to perform a full restore of their server locally. They saw the varying timestamps of the files, but didn't think anything of it at the time. However, after they tried to launch the application, several of the files turned out to be corrupted, because they were stored in multiple extents – file segments – and each extent was from a different point in time. They also found that the relationships between some database tables were completely mangled. In fact, we found two related tables where the parent (header) table was dated a week older than the child (detail) table, leaving over 300 sets of orphan records in the child table after the restore!

Needless to say, restoring the data from the cloud wasn't any better, because the core problem was that the database was *never backed up properly in the first place*. Because of this core problem, even though they could restore a bootable copy of the server in the cloud, the data stored thereon was just as useless from a database perspective.

So, after much consternation and discussion, we finally were able to determine that the ONLY usable backup would be the FIRST one immediately following the last system reboot. Although this was a full 5 weeks prior, the database would be the "closest" to actually being in sync, because all of the files would have been closed during the reboot process, and the file timestamps would be as close as they could ever be to each other.

Now I'm not one to blame Murphy's Law for things like this, but you all know what's coming by now, right? Because of the amount of data being backed up on all of the various servers being protected, there was not enough storage on the local appliance to go back that far. Further, the data retention in the cloud was set to only three weeks, so there was no way to restore the data from 5 weeks prior! Argh!

Now, with no backups, what other options were there? Go back to the old server? Oops – the cutover was 5 weeks prior to all of this excitement, and the old server was already obliterated, effectively eliminating that option. What about restoring a backup of the old server? They were long gone, too. The ONLY remaining option was to restore the mangled data to an off-line folder for reference purposes, and then create an entirely NEW database for the mission-critical application and start entering ALL of their system data over again from scratch. Ugh.

Validating Your Backups

All of this leads up to the real purpose behind this paper – validating your backups. But what does it mean to *validate a backup*?

In short, validating a backup is the process of ensuring that your restored data is complete and perfectly valid, ready for production use in case you ever had to do a full restore of your environment. However, like with most things in life, there are various degrees of validation, and your current validation process may only be giving you part of the picture. Let's look at some of the common validation processes:

- **Server Boot Screen**: Does the backup of your server boot properly? Some backup solutions include the ability to virtually boot the backup image and show you the logon screen as proof that the backup is complete. However, you should note that this ONLY means that the OS is complete and boots with no issues. It tells you NOTHING about the applications or data that may be on that server.
- **Checking File Timestamps**: What if you boot the server, login, and check the timestamps of a few critical database files? What about checking ALL database file dates? If you use this option, you clearly didn't learn anything from the above story! When files don't get new timestamps, this solution is meaningless.
- **Boot, Login and Application Launch**: While more complete than a simple boot screen, actually logging into the server and launching the application is also not really complete. Sure, it will tell you that the environment seems to function, but you could be looking at data from 6 months ago!
- **Complete Data Validation**: Yes, the only way to completely validate everything is to not only launch the application, but also run some simple reports or search for data that was entered shortly before the backup was taken. This is far more complicated than most people want to do, but it is really the ONLY viable way to validate your environment.

In short, this is a non-trivial exercise, but to ignore such an exercise is to do so at your own peril (or that of your data, anyway). You MUST allocate sufficient time to completely test your data restoration process in order to ensure that your backups are valid for what they are needed for. After all, you don't do backups to do backups – you do backups to do a **restore**!

PSQL-Specific Validation

If you have a backup solution that has not been tested yet, then you will definitely want to test it to ensure that you are getting clean backups. Of course, the best option is to follow the above Complete Data Validation process and prove to yourself that everything is good.

Luckily, there is an easier way. We have built a straightforward (if lengthy) testing process that will allow you to test your backup solution to confirm that it is properly backing up your PSQL data. You should follow these steps EXACTLY:

1. Create an empty directory on your server where it will be backed up normally.
2. Copy a medium-sized database file from your application into the new folder.
3. Wait for the next backup window and verify that the file is backed up. Note the timestamp on this file.
4. Open the PSQL Function Executor.
5. Using the File/Open dialog box, open the database file from the test directory.
6. Click the StepFirst icon () in the upper right corner to read the first record. Data from the first record will be displayed in the Data Buffer.
7. Modify one or more bytes in the Data Buffer section of the screen. You can change anything you want, such as putting in the current time so that you can identify it later on. (The changes won't break your application, because this is a private copy of your database file, of course.)
8. Click the Save icon () in the upper right corner to save the record. DO **NOT** CLOSE THE FUNTION EXECUTOR! Just let it sit there on the screen. This simulates a user running an application all day long and never signing out.
9. Check the timestamp of the file. Assuming the System Cache setting is OFF on your environment, you should see that although the data has been written to the file, the timestamp has NOT been changed.
10. Wait for the next backup window to complete.
11. Close the Function Executor.
12. Check the timestamp of the file – it should now be updated to the current time.
13. Wait for the next backup window to complete.

At this point, you have three separate backup snapshots that include this file. We'll call them A, B, and C. From your backup solution, restore the file from each of these instances in turn. (You may find it easiest to restore them each to a separate folder.)

When you look at the data files in the folders, you should see that file A has the original timestamp of the file that you copied into the directory. File B should have the same (original) timestamp. File C, however, has the new timestamp, which should be the time that you closed the data file.

It is now time for the moment of truth! Using the Function Executor, open each file in turn and read the first record:

- File A should contain the original, unchanged data record. Since this backup was taken before we did anything, this should not surprise you, and every backup package should get this one right.
- File C should contain the final, changed data record. Since this backup was taken AFTER we closed the file, the file's timestamp would have been updated, and the file should have been backed up normally. Again, every backup package should get this one right.

- What about File B? Remember that the data in this file was changed on the disk BEFORE the second backup, but since the file was not closed, the timestamp did NOT get updated. If you open the file, read the first record, and see the CHANGED data, then you can rest easy – your backup solution is capturing your database changes properly. However, if you open the file and see the ORIGINAL data, then you have a problem -- your backup solution is NOT backing up PSQL databases correctly, and you need to address this defect before a full system restore is ever needed, or you will be dealing with a similar massive loss of data.

Solutions and Workarounds

As indicated above, the PSQL database will, by default, only update file timestamps when a file is closed by all users. This can leave some "hot" files (i.e. those opened by an application 24x7) with dates that are very old when compared to other, less-used files. Further, this causes any incremental or differential backup solution that uses the file timestamps as a trigger to miss the file on multiple, successive backups, even though it may be continuously updated throughout the day.

There are, however, a few other solutions to consider:

1. **Change Your Backup Solution Settings**: You may be able to find a configuration setting within your backup solution that forces it to identify changed files more completely. For example, if the software supports using the Windows Change Journal (a.k.a. USN) instead of timestamps, then enabling this option might just resolve your issues quickly and easily. If you don't find anything, contact the solution developer and ask for help. Sometimes, these switches can be buried, mislabeled, or even hidden in the registry or some other configuration file.
2. **Force Users to Exit**: If you are not running a 24x7 operation and can afford a downtime window every night, then the simplest solution is to first request that all users exit the application before they leave for the day. Then, you can use the PSQL Monitor (or a tool like Goldstar Software's free **KillUser** tool) to kick out any remaining users immediately before your backup starts. Since all users are out, all files are therefore closed, and they all have current timestamps for backup. Of course, this doesn't work well if you are running a mission-critical system with a requirement for 24x7 access.
3. **Enable System Cache**: The PSQL setting "Use System Cache" directly controls whether the database engine will leverage the OS cache or not. By default, this is OFF, which provides for faster disk writes. (By bypassing the OS cache, the system gets substantially better throughput.) Leaving it off also prevents double-caching on database reads, where the OS caches the data AND the database engine caches the data – further wasting memory. If you enable the System Cache setting inside the engine, then all disk reads and writes are sent through the OS layer. Again, this causes double-caching of reads and slower writes, but this ALSO allows the OS to update the file timestamp on each and every disk write!

If your system is lightly loaded and you can afford the performance penalty, then enabling the System Cache is an easy way to fix this issue.

4. **Leverage Continuous Operations Mode**: Continuous Operations Mode is an old standby, available as a trusted backup solution since Btrieve 6.x was released in the early 1990's. This special mode "freezes" the files in a snapshot at the data file level, eliminating the need to use VSS or any other mechanism to get a clean backup. However, it also re-timestamps every data file, ensuring that other backup mechanisms will still work, too. You can enable ContOps mode through the BUTIL application or via the more-automated PSQL Backup Agent. (Backup Agent is a free add-on for PSQLv11 and newer Server engines, and can be purchased for the PSQL Workgroup Engine or older Server engines at an additional cost.) Using the two methods (Continuous Operations Mode and VSS Writer) together is neither recommended nor supported by Actian, so if you use this option, you should disable your use of VSS with your database backups. (You can still use it for your OS backups.) If you are interested in this solution, please read our other white paper on Proper Backups so that you understand the possible pitfalls that you must avoid, such as backing up the delta files.
5. **Update Timestamps with the Backup Agent**: Activating the Backup Agent for a short time has a very interesting side effect – it forces ALL of the opened files to be put into Continuous Operations Mode, which then gives them a new timestamp! The trick here is to issue the PVBACKUP –ON command, and then immediately issue the PVBACKUP –OFF command, so that ContOps mode has a chance to complete on every file *before* the VSS snapshot occurs. If you do this, then EVERY open file will get a new timestamp, and therefore every open file will be seen as "different" by the backup environment, whether they have been actually updated or not. This may cause your differential or incremental backups to include a lot more data files than you would normally expect to see being backed up, increasing backup system storage needs. However, you are at least guaranteed to get all of the changed files.
6. **Change Backup Solutions**: As you can see, each of the above solutions has its own down sides, and none of them may be what you really want to do. In that case, changing to a different backup solution altogether may be in order. There ARE backup solutions that work correctly with PSQL available!

If you still can't get it to work, contact Goldstar Software and let us work with you to help! We can provide assistance and ideas to make it possible to grab weekly off-site backups, nightly images, hourly snapshots, and even continuous, near-real-time replication – everything you need to keep your data safe and your business running, regardless of what happens in the big, scary world out there….